VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Komprese dat pomocí kontextové mapy

Data Compression Using Context Map

2011 Michal Vašinek

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2011 .....

Na tomto místě bych rád poděkoval svému vedoucímu práce panu Ing. Janu Platošovi, PhD. za to, že byl ochoten mi tuto práci vést a pomáhat při její tvorbě. Rád bych poděkoval svým rodičům a přátelům za trpělivost a vyjádřenou podporu.

Abstrakt

V dnešní době existuje celá řada kompresních algoritmů, některé se více zaměřují na efektivitu z hlediska času stráveného kompresí, jiné upřednostňují schopnost dosáhnout co nejlepšího kompresního poměru. Mezi algoritmy jsou značné rozdíly v principu, na jakém pracují. V této práci se zabývám možnostmi využití kontextové mapy při provádění komprese.

Klíčová slova: kompresní algoritmus, kontextová mapa, metadata, paket

Abstract

Nowadays many compression algorithms exist, some of them focus on effectivity from the point of time spend with compression, others prefer ability to reach the best compression ratio. Among algorithms there are big differences in the principle they work on. In my bachelors work I deal with possibilites of using context map at performing compression.

Keywords: compression algorithm, context map, metadata, packet

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

V rámci této bakalářské práce se zabývám návrhem jednoho, konkrétního typu kompresního algoritmu založeného na kontextových mapách. Cílem této práce bude popsat návrh algoritmu, jeho principu, dílčích částí subalgoritmů a zhodnotit jejich výsledek. Do psaní této práce jsem vstupoval s nápadem, zda je možné založit kompresi dat na nějaké formě jejich popisu pomocí metadat.

U zrodu této práce stála překvapivě jiná myšlenka, než tvorba kompresního algoritmu, byla to touha dopsat si plugin do svého oblíbeného IM komunikátoru, který by místo mne komunikoval s přáteli, zda zrovna půjdu ven, nebo ne. Plugin měl fungovat tak, že si projde veškerou komunikaci, kterou jsem v minulosti měl a vytvoří si vazby mezi jednotlivými částmi textu. Já bych mu poté jenom nastavil svůj status, jestli chci jít ven, nebo ne, a program by podle něj "vyhandlil"zda někam půjdu a dal mi vědět výsledek.

Když jsem začal psát tento plugin, začaly mne napadat další možnosti využití takovéto kontextové mapy. To byla chvíle, kdy mne také napadla myšlenka, zda by bylo možné na základě tohoto kontextu vytvořit nějakou smysluplnou množinu metadat, která by byla schopna jednotlivé věty jednoznačně popsat, zmenšit a posléze zase vrátit do původního tvaru. Započal jsem období experimentování, a když jsem uviděl, že zde potenciál je, rozhodl jsem se, že by stálo za to tento algoritmus analyzovat důkladněji v rámci bakalářské práce.

Postupně v této práci budu popisovat Shannonovu Teorii Informací v kapitole 2, dále se zaměřím na základní typy kompresních algoritmů a speciální pozornost budu věnovat v kapitole 3.2.4 na kontextu založeným metodám typu PPM, ze známých algoritmů je PPM nejblíže mému pojetí.

Další části budou obsahovat především konkrétní návrh algoritmu, analyzování kontextové mapy, jakých vlastností kontextová mapa nabývá, jak se vyvíjí její velikost, jak ovlivní výslednou kompresi, abych nakonec vše shrnul a porovnal se současnými kompresními algoritmy.

V sekci A přílohy práce se věnuji zápisu kontextové mapy pomocí matic s využitím "bra-ket"zápisu, sekce B potom obsahuje zkrácenou verzi návrhu kontextového kompresního algoritmu pro kompresi založené na menší jednotce, než slovo běžného lidského jazyka, na jednom znaku. V sekci C mám doplněny další grafy a výsledky nad odlišnou množinou testovacích dat.

2 Teorie informací

Pro mou práci je Teorie informací klíčovou teorií, proto se jí budu věnovat hned v úvodu práce a později se k jejím důsledkům budu vracet a využívat je v některých oblastech návrhu kontextového kompresního algoritmu. Tvůrcem teorie informací je americký matematik Claude Elwood Shannon[1]. Pojďme se nyní zaměřit na nejdůležitější body této teorie.

2.1 Entropie

Entropie je pojem přejatý ze statistické fyziky, sloužící k popisu množství informace obsažené v jednotlivých zprávách a k vyjádření pravděpodobnosti těchto zpráv.

Definice 2.1 *Zpráva je datový objekt určený ke kompresi*

Matematicky se entropie zapisuje následující rovnicí:

$$H(S) = \sum_{s \in S} p(s) log_2 \frac{1}{p(s)} \tag{1}$$

- $H(S)$ je entropie zpráv
- $p(s)$ je pravděpodobnost jedné určité zprávy s

Entropie je vážený průměr informací obsažených v každé zprávě, a tudíž průměrný počet bitů informace v množině zpráv.

Poznámka 2.1 Zařízení se dvěma stabilními stavy, jako je třeba flip-flop obvod, může uchovat jeden bit informace. N takových zařízení tak uchová 2^N stavů a zároveň platí, že $log_2 2^n = n$.

2.2 Vlastní informace

Uvažujme jednotlivé zprávy $s \in S$. Shannon definoval zápis vlastní informace nesené zprávou jako:

$$i(s) = log_2 \frac{1}{p(s)} \tag{2}$$

Takto definovaná vlastní informace reprezentuje počet bitů informace obsažené ve zprávě a zároveň počet bitů, který bychom měli použít, chceme li zprávu odeslat. Rovnice říká, že zprávy s vyšší pravděpodobností budou obsahovat méně informací.

2.3 Podmíněná entropie

Při utváření této práce jsem došel k závěru, že pravděpodobnost jednotlivých zpráv je založena na kontextu, ve kterém se daná zpráva vyskytuje. Kontext obecně snižuje entropii systému. V současnosti se používají dvě zajímavé kompresní metody založené na kontextu, JBIG a PPM. Blíže se budu zabývat v kapitole 3.2.4 metodě PPM.

Definice 2.2 *Za kontext se považuje okolí zkoumaných dat, kontextem ve smyslu metody PPM je myšleno několik předcházejích znaků, ve smyslu metody JBIG jsou kontextem myšleny pixely v okolí jednoho konkrétního pixelu. Kontextem ve smyslu kontextového kompresního algoritmu je množina slov a asociací sestavená algoritmem na základě konkrétní tématické oblasti.*

Podmíněná pravděpodobnost události e založené na kontextu c se značí jako $p(e|c)$. Celková pravděpodobnost události e je dána:

$$p(e) = \sum_{c \in C} p(c)p(e|c) \tag{3}$$

- C je množina všech dostupných kontextů

Zápis vlastní informace události e v kontextu c je v tomto případě dán vztahem:

$$i(e|c) = log_2 \frac{1}{p(e|c)} \tag{4}$$

Průměrná podmíněná vlastní informace je nazývána jako takzvaná podmíněná entropie zdrojových zpráv. Pro množinu zpráv S a kontext c je podmíněná entropie definována jako:

$$H(S|C) = \sum_{c \in C} p(c) \sum_{s \in S} p(s|c) log_2 \frac{1}{p(s|c)} \tag{5}$$

Z tohoto vztahu můžeme ukázat, že pokud je rozdělení pravděpodobnosti množiny S nezávislé na kontextu c , pak $H(S|C) = H(S)$, tudíž pomocí použití kontextu můžeme entropii systému pouze snížit.

3 Kompresní algoritmy

Od okamžiku vytvoření prvních koncepcí počítače začaly vznikat nejrůznější algoritmy pro zefektivnění jejich chodu. Jedním z úkolů té doby bylo, především z důvodu ceny paměti, zmenšit data ukládaná na paměťová uložiště.

V této kapitole se budu věnovat kompresním algoritmům, proto nejdříve nastíním hlavní problém týkající se komprese dat. Vezměme si nějakou jedno bajtovou zprávu, taková zpráva se skládá ze sekvence osmi bitů. Pokud si přejeme takovouto zprávu zkomprimovat, je naším cílem udělat tuto zprávu menší(ve smyslu použití méně bitů na její strojovou reprezentaci). Naše zpráva o velikosti jednoho bajtu může kódovat 2^8 zpráv, cílem kompresních algoritmů však je dosáhnout toho, aby výsledná zkomprimovaná zpráva měla velikost menší než je 8 bitů.

Z tohoto požadavku na kompresní algoritmus vyvstávají dvě důležité otázky:

1. Je možné zmenšit počet bitů, který potřebuje původní zpráva na menší počet bitů?
2. A pokud na první otázku je odpověď "ANO", jakým způsobem toho docílit?

Na první otázku je odpověď tato, možné to je, ale pouze za předpokladu, že zpráva, kterou si přejeme zkomprimovat, vychází z množiny zpráv, které mají vzájemně rozdílnou pravěpodobnost výskytu, tedy taková množina zpráv je nehomogenní.

Tuto definici je možné převrátit, žádná zpráva náležející do homogenní množiny zpráv(tedy takové množiny zpráv, kde každá zpráva se vyskytuje v množině se stejnou pravděpodobností) nemůže být zmenšena ve své velikosti.

Odpověď na druhou otázku je základem téměř všech současných kompresních algoritmů, tedy, že pokud zjistíme, že nějaká zpráva se vyskytuje s vyšší pravděpodobností, nebo třeba, že některé vzory se v určitém balíků dat vyskytují více, než jiné, lze toho využít pro rekonstrukci komprimované zprávy v tom smyslu, že některé hodnoty jsou pro danou pozici pravděpodobnější než jiné.

3.1 Kritéria pro vyhodnocení úspěšnosti kompresních algoritmů

3.1.1 Vyhodnocování bezeztrátových algoritmů

1. Čas potřebný pro kompresi
2. Čas potřebný pro dekompresi
3. Velikost zkomprimovaného souboru, kompresní poměr
4. Obecnost - je algoritmus stejně úspěšný při kompresi binární a textových souborů

3.1.2 Vyhodnocení ztrátových algoritmů

Vyhodnocení ztrátových algoritmů je o něco obtížnější, protože musíme brát v úvahu kvalitu aproximace, tato vlastnost je obzvláště důležitá při kompresi obrázků.

Jednou z běžně používaných metod porovnání algoritmů je od Jeffa Gilchrista nazvaná Archive Comparison Test. Tato metoda je zaměřena na porovnání času a výsledného kompresního poměru.

3.1.3 Calgary korpus

Calgary korpus je standardním nástrojem pro hodnocení(benchmark) kompresního poměru, převážně se skládá z anglického textu. Obsahuje dvě knihy, pět odborných článků, jednu bibliografii, kolekci novinových článků, tři programy, jeden výpis terminálového sezení, dva objektové soubory a jeden bitmapový obrázek.

3.2 Pravděpodobnostní kódování

Rozlišujeme mezi algoritmy, které přiřazují každé zprávě unikátní posloupnost bitů, a takové, které spojují kód dohromady z více než jen jedné zprávy. Prvními se budu zabývat Huffmanovými kódy, které jsou označovány jako tzv. "Prefix codes", později se budu zabývat aritmetickými kódy. Aritmetické kódy mohou dosáhnout lepšího kompresního poměru, ale je to vykoupeno větším časovým zatížením při kompresi a dekompresi.

3.2.1 Prefixové kódy(směrové kódy)

V oblasti počítačů obvykle pracujeme s kódy pevné délky, jako příklad může posloužit ASCII kód, ten přiřazuje každému tisknutelnému znaku a několika dalším řídícím znakům sekvenci sedmi bitů. Pro účely komprese by však bylo vhodnější mít kódovací slova, která mohou mít proměnnou délku v závislosti na pravděpodobnosti dané zprávy.

Variabilní délka slova však přináší potenciální problémy, pokud dekomprimujeme nějakou zprávu, potřebujeme znát, kde tato zpráva začíná a kde končí. Tyto dvě informace budou vyžadovat určitý paměťový prostor.

Efektivnějším řešením je navrhnout kódy, ze kterých můžeme vždy jednozančně dekódovat sekvenci bitů na svá kódová slova. Takové kódy se označují jako "jednoznačně dekódovatelné kódy".

Na směrový kód se nahlíží jako na binární strom, kde:

- Každá zpráva je listem stromu.
- Kód pro každou zprávu je dán následováním cesty od kořene stromu k listu a přidáním nuly pokaždé, když přecházíme do levé větve, respektive přídáním jedničky pokaždé, když přecházíme do pravé větve.

3.2.2 Huffmanovo kódování

Huffmanův kód je optimální prefixový kód vytvořený z množiny pravděpodobností daných Huffmanovým kódovacím algoritmem. David Huffman vytvořil tento algoritmus při studiích informačních teorií na MIT v roce 1950. Algoritmus je dodnes používanou součástí mnoha algoritmů, slouží jako základ pro komprese GZIP, JPEG a několika dalších.

3.2.3 Aritmetické kódování

Aritmetické kódování je metodou pro dosažení bezeztrátové komprese na základě entropického kódování s proměnnou délkou kódového slova. Zjednodušeně řečeno, pokud komprimujeme pomocí aritmetického kódování přiřazujeme znakům, které se vyskytují častěji menší počet bitů, zatímco ty, které se vyskytují zřídka budeme kódovat větším počtem bitů. Principiálně vyžaduje úspěšnost této metody nehomogenní rozdělení pravděpodobnosti zpráv.

3.2.4 PPM

Příkladem kompresního algoritmu založeném na kontextu je kompresní algoritmus PPM. Metoda byl navržena v práci J. Cleara a I. Wittena[4] a implementována v práci A. Moffata[5]. Algoritmus je založen na principu kodéru, který uržuje statistický model textu.

Hlavní myšlenkou algoritmu je přiřadit nějakému symbolu S pravděpodobnost P nejenom na základě frekvence jeho výskytu v textu, ale také na kontextu, ve kterém se vyskytuje. Podívejme se nyní na dva přístupy ke tvorbě modelu.

Statický kontextově založený model používá vždy ty samé pravděpodobnosti. Obsahuje statickou tabulku s pravděpodobnostmi všech možných dvojic, trojic, případně větších n-tic znaků abecedy a používá tuto tabulku k přiřazení pravděpodobnosti, že každý následující symbol S záleží na předcházejícím symbolu C. Tabulku lze sestavit na základě dostatečného počtu testovacích dat. Takovýto model přináší dva problémy. Zaprvé platí, že některé vstupní řetězce mohou být statisticky velmi rozdílné oproti původní tabulce, což v důsledku přinese nárůst potřebné paměti pro její uchování. Druhým problémem jsou potenciální nulové pravděpodobnosti u některých kombinací n-tic symbolů.

Adaptivní kontextově založený model podobně jako statický model uchovává tabulku n-tic symbolů abecedy a pužívá tabulky k přiřazení pravděpodobnosti, že nadcházející symbol S závisí na několika předcházejících symbolech. Tabulka je aktualizována vždy když se na vstupu algoritmu objeví další data. Takový model je pomalejší a komplexnější

než statický, ale poskytuje lepší kompresní poměr, dokonce i v případech, kdy vstupní data mají velmi rozdílné rozložení pravděpodobnosti oproti průměrnému výskytu.

Adaptivní kontextově založený model N-tého řádu načítá symbol S ze vstupu a zároveň vyhodnocuje N symbolů, které mu předcházejí. Takže například pokud bude algoritmus 20-řádu, bude odhadovat pravděpodobnost P, že se S objeví ve vstupních datech na základě kontextu o 20 symbolech. Teoreticky platí, že čím vyšší N, tím je také vyšší schopnost algoritmu přesně predikovat další znak. Nicméně vysoké hodnoty N vedou ke třem problémům:

1. Pro vysoké N(myšleno tisíce a více symbolů) vyvstává problém kódování první sekvence znaků. řešením zůstává možnost tyto symboly nekódovat a ukládat je jako prostý ASCII text, nebo v původní binární podobě.

2. Vysoká N vedou k velkému počtu dostupných kontextů(sekvence symbolů), například pro N=1, dostáváme pro 1-bytový symbol velikost kontextu rovnu $|c| = 256$, tedy obecně je velikost kontextu dána vztahem: $|c| = 256^N$.

3. Pokud používáme delší kontexty, uchováváme v tabulce zároveň více informací nashromážděných ze starších dat(myšleno ve smyslu vstupních dat, například pokud bychom algoritmu předali tuto bakálářskou práci, on by si nejdříve začal tvořit tabulku n-tic na základě úvodu a teprve poté by přešel k odbornějším výrazům). Lepších kompresí lze dosáhnout, pokud starší data mají menší důležitost, než-li data "čerstvá" .

Vzhledem k výše popsaným omezením se v praxi využívá rozsah N mezi 2 až 10.

Kodér PPM využívá jednoho zajímavého algoritmu. Vždy, když kodér nenalezne ve svém modelu kontext o délce N následovaný symbolem S, vyzkouší, zda lépe nebude vyhovovat kontext o délce N-1, tímto způsobem algoritmus zmenšuje velikost kontextu, dokud nenalezne kontext s nenulovou pravděpobností, že bude následován výskytem symbolu S.

Ve větším detailu algoritmus pracuje následujícím způsobem. Kodér si načte další symbol ze vstupu, vezme předcházejících N symbolů(neboli kontext N-tého řádu) a na základě původních dat, kterými již prošel, určí pravděpodobnost P, že se symbol S objeví za kontextem C. Kodér poté volá algoritmus pro adaptivní aritmetické kódování a zakóduje symbol S s pravděpodobností P.

Pokud kodér PPM(v tomto konkrétním případě PPMC) na určitý symbol S dosud nenarazil, přiřadí mu pravděpobnost rovnu $1/M$,kde M je počet prvků abecedy. V jiném případě, pokud kodér při snižování řádu kontextu neuspěje s určením nenulové pravděpobnosti, tedy dojde ke kontextu řádu 0, pak pro zjištění pravděpodobnosti P vyhledá počet výskytů symbolu S a vydělí jej počtem symbolů, které dosud přečetl.

PPM dekodér funguje odlišně než-li kodér. Zatímco kodér ví exaktně, který symbol je tím následujícím, dekodér musí zjistit, který symbol to byl. Problém je, jak zjistit, kdy se kodér rozhodl přejít ke kratšímu kontextu. PPM pro vyřešení tohoto úskalí využívá únikového symbolu("escape symbol"). Kdykoli kodér rozhodne o přejití ke kratšímu kontextu, nejdříve zapíše na výstup tento únikový symbol. Dekodér poté může přečíst tento únikový symbol a na jeho základě se také přepne do kontextu nižšího řádu.

Metoda PPM má několik variant, výše jsem zmínil metodu PPMC, další varianty jsou PPMA,PPMB,PPMP,PPMX a PPMd. Hlavní rozdíl mezi nimi je v procesu zpracování a ohodnocení pravděpodobnosti únikových symbolů.

Metoda PPMA přiřazuje únikovému symbolu pravděpodobnost $1/(n+1)$,což je ekvivalentní přiřazení v kontextu řádu 1. Ostatním symbolům je přiřazena původní pravděpodobnost x/n a součet všech těchto pravděpodobností dává v součtu 1(úniková sekvence se nepočítá).

Metoda PPMC přiřazuje pravděpodobnost symbolu S následujícím kontext C pouze, pokud se vyskytoval v předchozím kontextu dvakrát. Pokud byl výskyt menší než 2, pak takovému symbolu žádnou výslednou pravděpodobnost nepřiřazuje.

PPMP je založena na předpokladu, že se symbol S objeví podle Poissonova rozložení s očekávanou hodnotou λ_S. Obdobně přiřazuje pravděpodobnosti také metoda PPMX.

4 Kontextový kompresní algoritmus

Algoritmus, který bych touto prací rád představil, je kompresním algoritmem založeným na pravděpodobnosti a jejím rozložení v rámci sítě vzájemně propojených prvků, které reprezentují jednotlivé informační jednotky(IU).

Definice 4.1 *Jednotka informace(IU) je atomická informace, může jí být jedno slovo reálného(např. českého či anglického) jazyka, stejně tak to může být pouze jeden znak, či skupina znaků.*

Původ algoritmu částečně vychází z pozorování učiněných v rámci kvantové fyziky, a to v těchto bodech:

- Systém se může vyvíjet všemi cestami, ale každá cesta má jinou pravděpodobnost.
- Částice si otestuje při svém pohybu všechny cesty, ale vydá se právě jednou správnou cestou.

Samotný kompresní algoritmus je sadou menších algoritmů zajišťujících celkové fungování programu. Tyto algoritmy řeší tři typy úloh:

1. Budování kontextové mapy cest, popisující závislosti mezi informačními jednotkami.
2. Popsání cesty vkládané do datové jednotky procházející mapou.
3. Nalezení původní cesty na základě popisu obsaženého v datové jednotce.

Budování kontextové mapy je analogické k problému procházení louky s vysokou trávou, pokud vstupujeme na louku poprvé a vydáme se libovolnou cestou, zanecháme za sebou ušlapanou cestičku, čím více-krát touto cestičkou jdeme, tím více bude vychozená. Stejně tak po louce může jít někdo jiný na úplně jiné místo, a ještě třeba několik dalších lidí. Když potom přijde cizí člověk, vidí před sebou několik cest různě vyšlapaných, může se vydat cestou nejprošlapanější, ale pokud se třeba s někým poradil, tak se může řídit jeho malou radou a vydat se jinou cestou. Účelem algoritmů budujících mapu je vytvoření takové sítě cest, aby v nich každá sekvence zpráv, s alespoň minimální pravděpodobností výskytu, našla své řešení.

Popsání cesty je dalším úkolem, který je třeba řešit. Pokud se vrátím k analogii louky s vysokou trávou, tak k popisu cesty ve svém algoritmu používám místa, kde se potenciálně dvě nebo více cest překříží, každé takové místo reprezentuje informační jednotku. Informační jednotka je zástupným symbolem pro specifickou sekvenci bitů. K tomuto problému přistupuji v odlišných verzích algoritmu různě. V prvním případě je sekvencí bitů slovo skutečného lidského jazyka a ve druhém případě jsem posunul míru abstrakce k jedno bytové informační jednotce, která ve strojovém světě může reprezentovat 256 hodnot.

Dříve jsem zmínil pojem datové jednotky procházející mapou, tato jednotka slouží k tomu, aby si zaznamenávala informace o cestě, kterou prochází. Úkolem takové jednotky je posbírat co nejrelevantnější metadata o cestě tak, aby z nich bylo možné zpětně tuto cestu, po které se datová jednotka vydala rekonstruovat.

4.1 Komprese pomocí slovníku slov lidského jazyka

První verze algoritmu je postavena nad slovy lidského jazyka. Mou snahou je v tomto algoritmu vybudovat takovou mapu, která by při použití datových jednotek pro průchod mapu dokázala zmenšit obsah člověkem psaného textu.

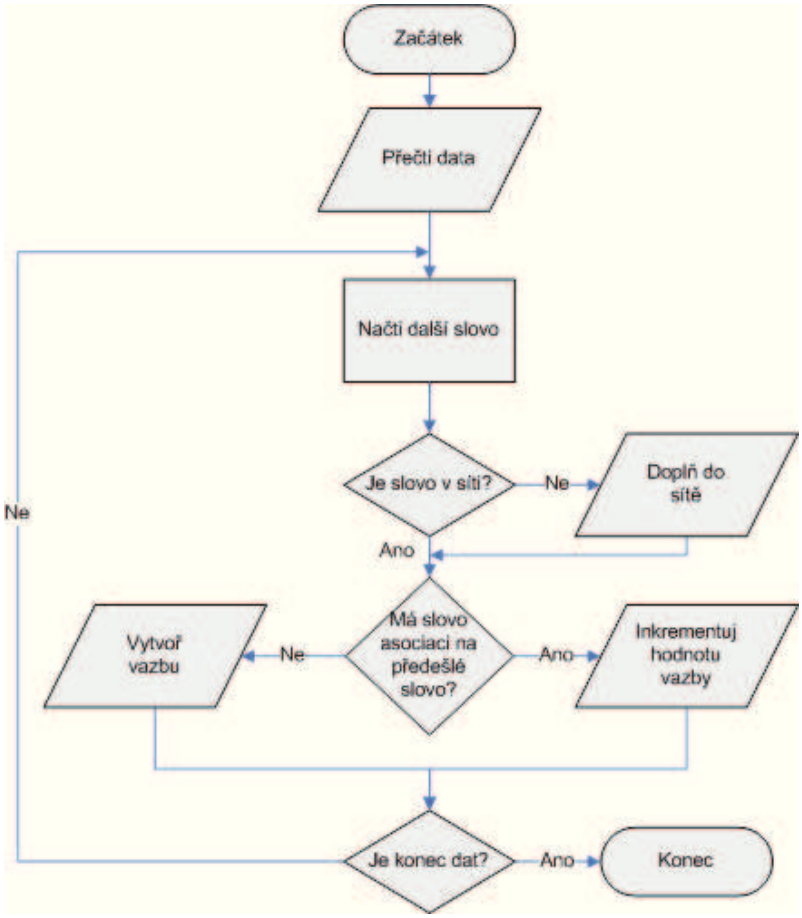
Definice 4.2 *Slovo ve smyslu kontextové mapy je posloupnost symbolů zvolené abecedy, kde platí, že vždy když provádíme konkatenaci dvou slov, výsledné slovo je dáno spojením prvního slova, symbolu reprezentujícím mezeru a druhého slova, v tomto pořadí.*

4.1.1 Algoritmus pro vybudování kontextové mapy

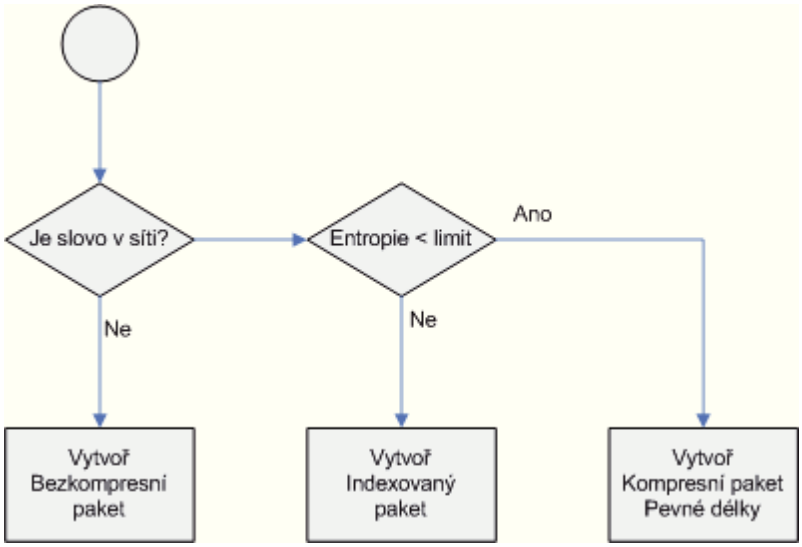
Budování mapy je založeno na principu vkládání vět lidského jazyka slovo po slově do mapy a budování vzájemných vazeb mezi nimi, tato mapy je ohodnoceným orientovaným grafem, kde každým vrcholem je jedno slovo a každá hrana je ohodnocena počtem průchodů touto hranou, vývojový diagram algoritmu je zakreslen na obrázku 1:

1. Rozdělení věty na jednotlivá slova, tak ať jsou seřazena od prvního k poslednímu slovu věty
2. Věta se prochází sekvenčně po jednotlivých slovech a pro každé slovo se provede kontrola, zda již v mapě neexistuje, pokud slovo v mapěi není, je do mapy doplněno
3. Nyní je slovo jednoznačně v mapě, pokud danému slovu předcházelo jiné slovo, vazba mezi těmito slovy se inkrementuje(v případě, že žádná vazba mezi těmito slovy dosud není, vytvoří se), pokud se před slovem žádné slovo nenacházelo, pokračuje se dále
4. Každé slovo se uloží, aby v pořadí další slovo, znalo slovo předcházející
5. Body 2,3 a 4 se opakují přes celý zdrojový soubor dat

Na konci sběru dat je vybudována mapa s asociacemi mezi jednotlivými slovy, které v podstatě pouze počítají absolutní výskyt slova. Důležitá je v tomto případě především myšlenka, že slova s nejvyššími hodnotami výskytu se budeme snažit eliminovat a naopak budeme hledat vazby s menším a středním počtem výskytů, které budou vhodné pro kompresní algoritmus.



Obrázek 1: Vývojový diagram utváření kontextové mapy



Obrázek 2: Diagram výběru paketu

4.1.2 Algoritmus síťového paketu

V síti rozlišujeme tři typy paketů:

1. Bezkompresní paket
2. Indexovaný paket
3. Paket fixní velikosti

Princip jakým se vybírá příslušný paket je zobrazen na obrázku.

Definice 4.3 *Bezkompresní paket je při kompresi používán pokud právě zpracovávané slovo nebylo v síti nalezeno.*

Definice 4.4 *Indexovaný paket se v síti používá, pokud slovo bylo nalezeno, ale nebyla nalezena asociace na další slovo v pořadí a nebo v případě, když má slovo příliš vysokou entropii, tzn. vyšší než mezní nastavenou entropii.*

Definice 4.5 *Paket fixní velikosti se v síti používá, pro nalezení metadat a je klíčovým paketem pro provádění komprese a dekomprese.*

4.1.3 Paket fixní velikosti

Komprese a dekomprese je založena na konceptu síťového paketu. Tento paket je transportní jednotkou, která prochází sítí a zaznamenává si informace o průchodu každým slovem. Je to sběrač metadat o cestě, kterou prošel, v případě kompresního algoritmu jsou to metadata o datech, která si přejeme zkomprimovat.

0-1 bit	2-7 bit	1-2 byte	3-4 byte	5-6 byte	7-8 byte
T	D	FI	LI	CHK1	CHK2

- T - Typ paketu(00 - Bezkompresní paket, 01 - Indexovaný paket, 10 - Paket fixní velikosti
- D - Počet slov, které komprimujeme, z faktu, že jsem vyhradil pro uchování délky paketu 6 bitů vyplývá, že nejvyšší počet slov, která můžeme do paketu vložit je 64
- FI - Index prvního komprimovaného slova, vyhrazeny 2 bajty, maximální velikost 2^{16}
- LI - Index posledního komprimovaného slova, vyhrazeny 2 bajty, maximální velikost 2^{16}
- CHK1 - Kontrolní součet, vyhrazeny 2 bajty, maximální velikost 2^{16}
- CHK2 - Střídavý kontrolní součet, vyhrazeny 2 bajty, maximální velikost 2^{16}

Tabulka 1: Zápis bitů(bytů) fixního kompresního paketu

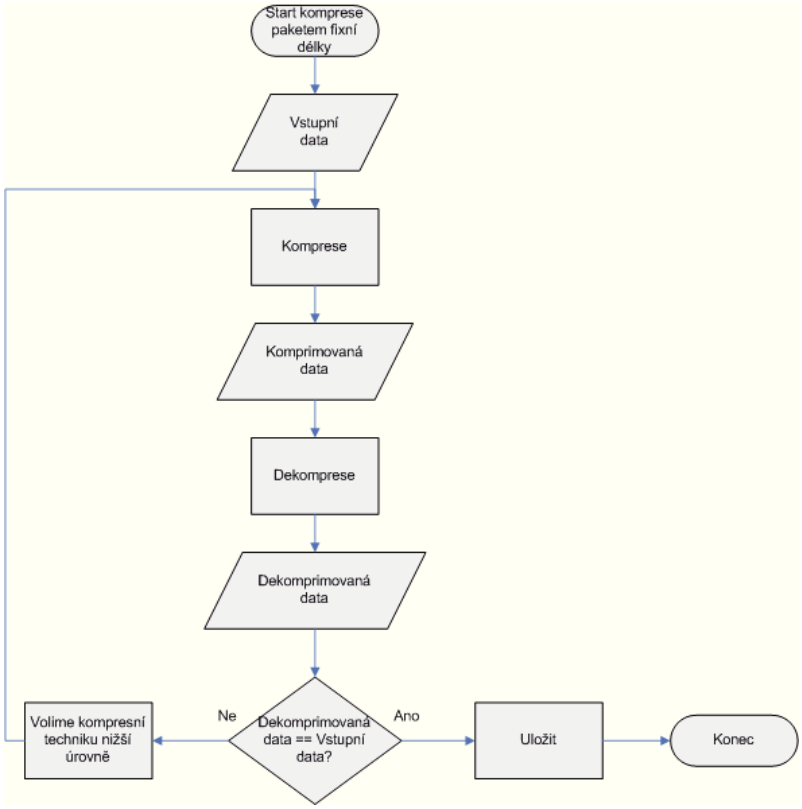
Předpokládejme, že máme vybudovánu mapu, tak jak jsem koncept představil v předchozí podkapitole. Tomuto síťovému paketu předáme pole s určitým počtem slov, který si v algoritmu nadefinujeme. Síťový paket vpustíme do sítě, aby si mohl posbírat metadata a ta na závěr uložíme.

Síťový paket pro své správné fungování potřebuje znát určitá metadata těmito metadaty jsou:

- Index prvního slova zprávy
- Index posledního slova zprávy
- Počet slov ve zprávě
- Kontrolní součet
- Pro optimalizaci také maximální zanoření v jednotlivých slovech
- Při kompresi dat je v paketu uloženo pole slov ke kompresi

Výsledná podoba paketu je zobrazena v tabulce 6:
Nyní přejděme k postupu sestavení zkomprimované zprávy:

1. Síťový paket si zapíše index prvního slova, algoritmus potřebuje znát místo odkud do sítě vstoupit, pokud by tato hodnota algoritmu nebyla dána, musel by při dekompresi procházet všechna dostupná řešení, což není výpočetně realizovatelné.



Obrázek 3: Diagram komprese fixním paketem

2. Nyní síťový paket prohledá možné cesty(spojení k jiným slovům), zda se tam vyskytuje, či přesněji na jaké pozici se vyskytuje následující slovo. Pokud je toto slovo slovem s nejsilnější vazbou, tedy má nejvíce ohodnocené spojení, pak se považuje úroveň zanoření za nula, pokud je toto slovo druhé v pořadí podle ohodnocení vazby od předchozího slova, nastaví se maximální úroveň zanoření v paketu na jedna atd. .

3. V dalším kroku se síťový paket vydá k dalšímu slovu, na toto slovo musí být vytvořeno spojení z prvního(předchozího) slova, do paketu se přičte k hodnotě kontrolního součtu hodnota vazby mezi těmito slovy.

4. Síťový paket takto cestuje sítí, dokud nedojde k poslednímu slovu.

5. Po dokončení cesty se metadata ze síťového paketu uloží a pokračuje se další zprávou.

4.1.4 Algoritmus dekomprese pomocí síťového packetu

Dekomprese probíhá obdobně jako komprese, jenomže v opačném pořadí. Tentokrát je úkolem síťového paketu hledání takové cesty, která bude odpovídat metadatům, která si sebou nese. A to především kontrolního součtu, počtu slov a odpovídajícímu prvnímu a poslednímu slovu.

Princip hledání správného řešení je ten, že cesta, která bude souhlasit s metadaty, je ta správná cesta. Postup dekomprese je následující:

1. Síťový paket vstoupí do sítě v bodě reprezentujícím slovo s indexem prvního slova v metadatech.

2. Od tohoto slova začíná hledání řešení. Pravidlo hledání cesty je potom následující. Začíná se cestou ke slovu, které má nejvýše ohodnocené spojení, až se dojde k poslednímu slovu, fakt, že slovo je poslední vychází z toho, že síťový paket si počítá průchody jednotlivými slovy a že známe celkový počet slov, která jsou ve zprávě zakódována. Jako první cesta se vždy volí ta s nejvyšším ohodnocením.

3. Pokud síťový paket procházející sítí nevyhoví podmínkám, to jest: přesáhne počet slov, přesáhne kontrolní součet, nemá odpovídající index posledního slova, v takovém případě je paket zahozen. Testování na platnost paketu je prováděno při každém průchodu paketu slovem.

4. Paket se správným, tedy původním řešením, musí splnit podmínky kontrolního součtu, počtu slov a prvního a posledního indexu.

4.1.5 Analýza vlastností kontextové mapy

Pro co nejefektivnější fungování algoritmu je klíčovým faktorem univerzální, obecná úspěšnost kontextové mapy při kompesi. Je tedy důležité zkoumat vlastnosti kontextové mapy a jak tyto vlastnosti ovlivňují výsledek komprese. Jaké vlastnosti to jsou?

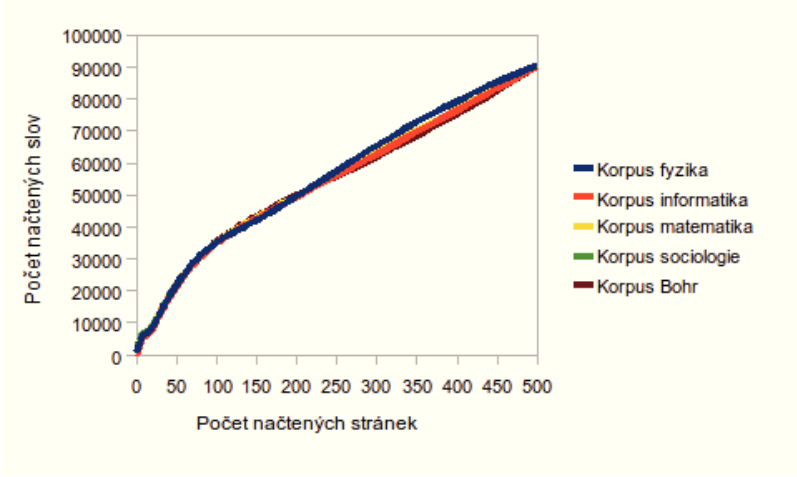
1. Počet stránek, který je zapotřebí načíst. Je výhodné tvořit rozsáhlou kontextovou mapu? Jaká je výtěžnost slov a asociací s každou další načtenou stránkou?

2. Úspěšnost komprese. Do jaké míry je rozsah kontextové mapy určujícím pro úspěšnost komprese? Jaká je závislost kompresního poměru vůči počtu načtených stránek, slov a asociací?

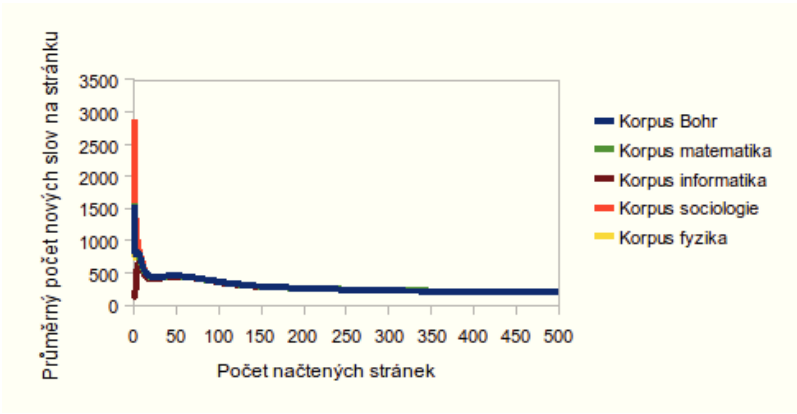
3. Nejvyšší míra nastavené entropie ve fázi komprese. Jak se mění závislost kompresního poměru vůči maximální povolené entropii pro kompresi pomocí fixního paketu?

Vlastnosti vypsané výše společně s odpovědmi na otázky u nich doplněných nám dají představu o tom, jak se vyvíjí kompresní vlastnosti algoritmu v závislosti na vlastnostech kontextové mapy.

Začněme se základním úkolem, vezměme si libovolný zdrojový soubor dat obsahující text v jazyce, nad kterým budeme kompresi provádět, v tomto případě jazyk anglický



Obrázek 4: Graf závislosti načtených slov na počtu načtených stránek

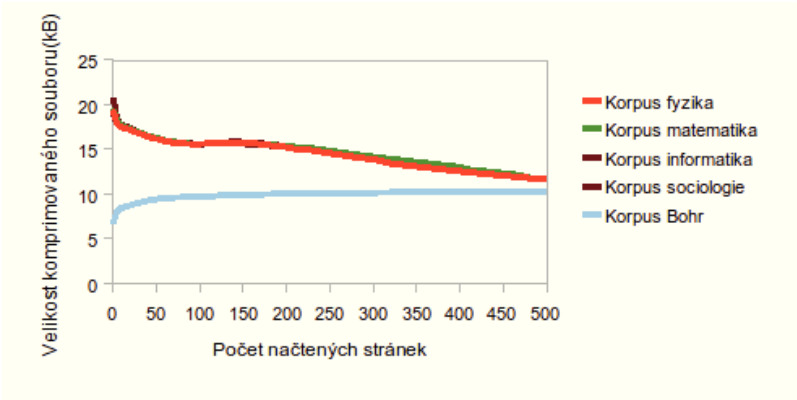


Obrázek 5: Graf závislosti průměrného počtu přidaných slov na počtu načtených stránek

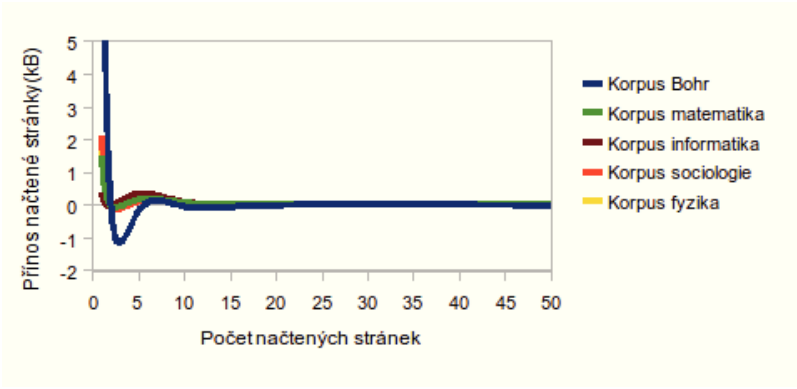
a za zdrojový soubor dat jsem zvolil článek o Nielsi Bohrovi. Pomocí tohoto souboru vytvoříme kontextovou mapu a skrze tento korpus zkomprimujeme tento soubor. Takovýto korpus označme termínem “vlastní korpus”. Tímto způsobem vygenerujeme základní(suboptimální) řešení. Toto řešení je charakteristické tím, že zdrojem je nejvýhodnější kontextová mapa pro daný soubor.

Výtěžnost slov a asoiciací při každé další načtené stránce nám dává závislost z Obrázků 4,8 a na Obrázku 5 potom můžeme vidět, jak se vyvíjí průměrný přírůstek slov s každou další přidanou stránkou. Pokud grafy porovnáme, zjistíme, že přibližně v oblasti, kde je načteno 100 stránek dochází k ustálení závislosti. Od tohoto bodu počet nových slov v kontextové mapě přibývá s každou načtenou stránkou konstantním tempem.

Na Obrázku 6 je zobrazena závislost velikosti výsledného komprimovaného souboru na počtu načtených stránek pro několik různých, tématických oblastí. Protože testovacím souborem byl článek o Nielsi Bohrovi, je z grafu patrné, že korpus sestavený na základě



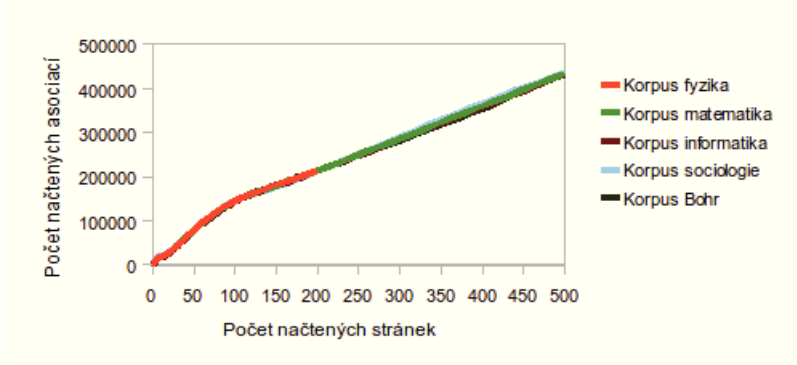
Obrázek 6: Graf závislosti velikosti komprimovaného souboru na počtu načtených stránek



Obrázek 7: Graf závislosti přínosu ke kompresi každé další stránky na počtu načtených stránek

tohoto článku(suboptimální korpus) je nejúspěšnější a s každou další načtenou stránkou se jeho schopnost úspěšně komprimovat snižuje. Oproti tomu u dalších korpusů dochází k opačnému jevu, s každou další načtenou stránkou kontextová mapa zlepšuje výsledek komprese. Obě křivky míří ke stejné hodnotě při vysokém počtu(více, než 500) načtených stránek. Z toho plyne, že pokud máme kontextovou mapu s dostatečně vysokým počtem slov a asociací, nezáleží pak na tématickém korpusu, ze kterého jsme tuto mapu sestavovali.

Pro názornost zde ještě uvedu graf přínosu ke kompresi každé další načtené stránky na počtu načtených stránek z obrázku 7. Graf přehledně ukazuje, že pokud jsem použil vlastní korpus testovacích dat, s každou další přidanou stránkou snižujeme kompresní schopnost mapy(záporná část na křivky Korpus Bohr), naopak opačný průběh u ostatních korpusů.



Obrázek 8: Graf závislosti počtu načtených asociací na počtu načtených stránek

4.1.6 Entropie mapy slov

Definici entropie, kterou použil Shannon, lze výhodně využít pro stanovení podmínek, za kterých má smysl nad jedním konkrétním slovem provádět kompresi. Můžeme kupříkladu stanovit přípustnou hodnotu entropie, při které ještě povolíme provedení komprese. Tímto způsobem algoritmus sám ošetří slova, která mají mnoho návazných asociací, a tudíž rozdělení pravděpodobnosti takové, že nemusí nutně dát výsledek v přijatelném čase. Tedy vazby, které směřují od slov s častým výskytem, se budu snažit eliminovat.

Nad jedním slovem mapy lze entropii s použitím Shannona definovat následovně:

$$H(slovo) = \sum_{a \in Asociace} p(a) log_2 \frac{1}{p(a)} \tag{6}$$

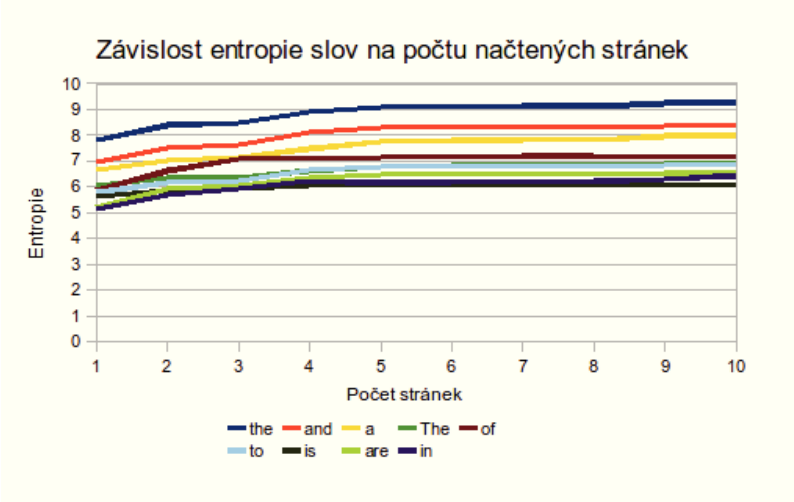
Jak jsem zmínil výše, pokud pomocí této rovnice budeme počítat entropii, můžeme ji omezit na určitou hodnotu, díky které eliminujeme slova jako mluvnické spojky, v angličtině například určité a neurčité členy a obecně všechna slova, která mají menší počet vazeb, nebo tyto vazby mají nerovnoměrné rozložení pravděpodobnosti, a tím jsou výhodné pro kompresi.

Obecně lze říci, že čím menší entropii asociací každé jednotlivé slovo má, tím je jednodušší a zároveň rychlejší odhalit při fázi dekomprese správné řešení.

Abych mohl správně analyzovat entropii mapy, vytvořil jsem si statistický přehled jednotlivých slov a jejich entropie, při různém počtu načtených stránek z encyklopedie Wikipedia.com . Přehled jsem prováděl nad anglickým jazykem s cílem, abych nalezl slova s extrémně vysokou entropií a udělal si obrázek o tom, která z nich nejsou vhodná pro kompresi.

4.1.7 Faktory ovlivňující kompresi

V rámci analýzy kontextu jsem stanovil tři faktory, které ovlivňují výslednou kompresi. Skrze tyto faktory je možné ovlivnit velikost zkomprimovaného souboru a čas potřebný pro dekompresi.



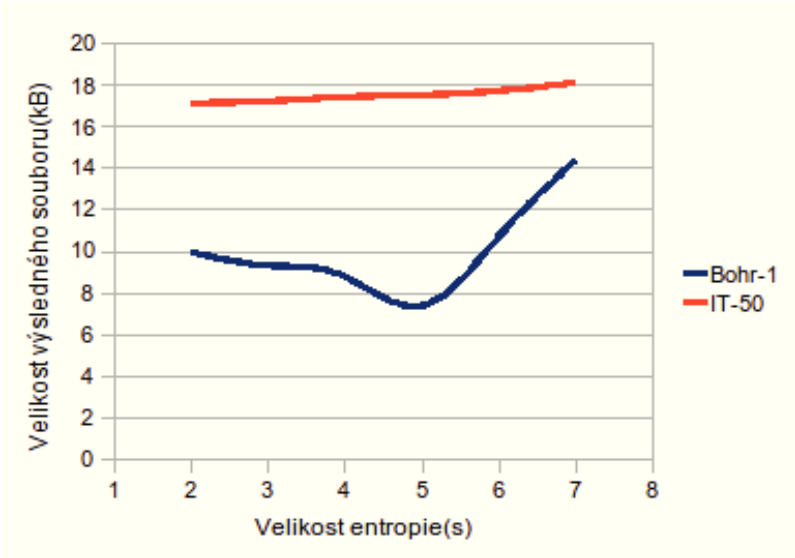
Obrázek 9: Graf závislosti růstu entropie na počtu načtených stránek

- Limitní počet slov komprimovaných fixním paketem
- Velikost mezní entropie asociací
- Časové razítko fixního paketu

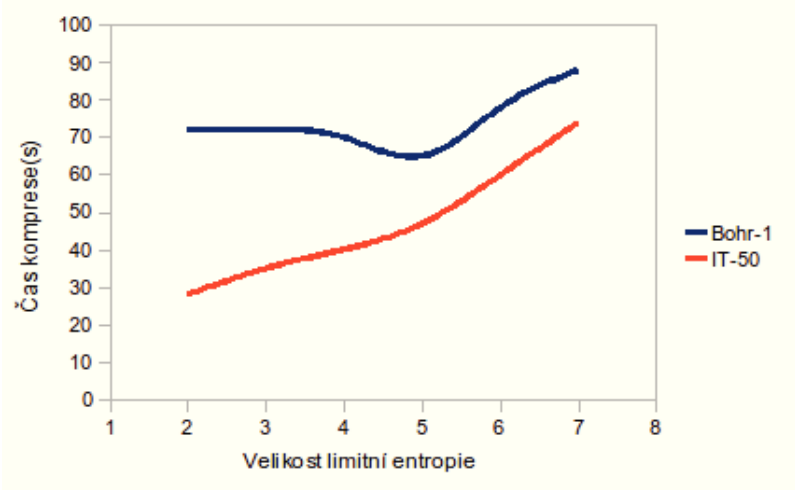
Limitní počet slov v kompresním paketu ovlivňuje jednak výkon, ale také úspěšnost algoritmu. Pokud při vkládání slov do kompresního paketu vložíme mnoho slov(velký počet vstupů), výrazně tím ovlivníme rychlost algoritmu. S každým dalším slovem totiž přibývá počet dostupných řešení, které musíme při dekompresi projít. Zároveň platí, že čím menší počet slov budeme komprimovat, tím rychlejší dekomprese dosáhneme. Oproti tomu vliv na kompresní poměr funguje obráceně, pokud volíme vyší limitní počet slov, algoritmus bude schopen komprimovat s lepším kompresním poměrem, naopak při nižším počtu slov se bude kompresní poměr zhoršovat.

Velikost mezní entropie asociací ovlivňuje především fázi komprese, ale ve svém důsledku výrazně přispívá k rychlosti při dekompresi. Podle entropie asociací se určí, která slova budou ještě do komprese zahrnuta. Tímto algoritmus dokáže úspěšně eliminovat slova, která by znamenala navýšení výpočetní náročnosti při dekompresi. Mezní entropie, stejně jako limitní počet slov ovlivňuje výsledný kompresní poměr. Na základě vzájemných vztahů, jsem stanovil základní hodnotu entropie na 5.

Časové razítko fixního paketu slouží především jako kontrola při fázi komprese, aby se eliminovala komprese posloupnosti slov, která i přes omezení v podobě limitního počtu slov a omezené entropie vede na časově náročné hledání správného řešení v kontextové mapě. Časovým razítkem stanovíme maximální přípustný čas pro dekompresi jedné posloupnosti slov.



Obrázek 10: Graf závislosti velikosti komprimovaného souboru na stanovené entropii



Obrázek 11: Graf závislosti času pro dekompresi na stanovené entropii

Program/algoritmus	Velikost po kompresi	Velikost po dvojité kompresi
ZIP	8,8 kB	
BZ2	7,8 kB	
LZMA	8,1 kB	
Algoritmus(Bohr-1)	7,1 kB	5,5 kB(ZIP)
Algoritmus(Fyzika-50)	9,2 kB	7,6 kB(ZIP)
Algoritmus(Uni-1000)	10,4 kB	8,3 kB(ZIP)

Tabulka 2: Porovnání výsledků s dalšími algoritmy

4.1.8 Problémy

Při tvorbě tohoto algoritmu jsem narazil na několik závažných problémů, převážně se týkaly rychlosti a velikosti uložené mapy:

- Příliš velký soubor obsahující v sobě vazby mezi slovy.
- Vysoké nároky na CPU při načítání kontextové mapy.
- Značné problémy algoritmu při slovech, která mají mnoho asociací, běžným příkladem mohou být například mluvnické spojky, předložky, přídavná jména, u těchto slov dochází k velkému rozptylu spojení a značně narušují výkonnost algoritmu.
- Pro každý jazyk je zapotřebí stanovit vlastní kontextovou mapu.

4.1.9 Shrnutí

V předešlých odstavcích jsem analýzu prováděl nad testovacím článkem o Nielsi Bohrovi, tento článek má velikost 20,6 kB. Testované kontextové mapy byly postaveny nad různými tématicky ucelenými oblastmi, jako například Informatika,Matematika, Sociologie, Fyzika. Výsledky dalších testovacích článků jsou uvedeny v Příloze C.

Porovnání výsledků s jinými kompresními algoritmy, při velikosti zvolené limitní entropie 5.0, maximálním počtu komprimovaných slov 8 je zobrazeno v Tabulce 2. Protože od vyšších počtů(> 500) načtených stránek dochází ke sjednocení kompresních vlastností kontextové mapy, poslední uvedený algoritmus je označen jako "Uni-1000".

4.2 Popis implementace

Algoritmus jsem implementoval jako konzolovou aplikaci ve VC++ v prostředí .NET. Pro .NET jsem se rozhodl především z důvodu rozsahu knihoven a nástrojů, které nabízí. V implementaci jsem potřeboval rychle a jednoduše stahovat stránky z Wikipedie, kde je mocnou knihovnou WebClient. Dále .NET usnadňuje práci s řetezci v různých kódováních, což je neocenitelné právě na příkladu Wikipedie, kde jsem se několikrát setkal s problémem při čtení arabských či řeckých znaků. Samotná aplikace je rozdělena do několika částí:

- UI část, systém menu a podmenu pro ovládání vlastností kontextové mapy, vlastností kodéru a dekodéru
- Třída reprezentující kontextovou mapu
- Třída reprezentující kodér a dekodér
- Třída reprezentující jedno slovo, tzv. neuron
- Třída reprezentující obecný paket, ze které potom dědí vlastnosti další odvozené pakety
- Několik pomocných tříd, pro generování statistik, pro stahování textů z Wikipedie, pro parsování webových stránek, pro ukládání kontextové mapy,...

4.2.1 Kontextová mapa

Z pohledu implementace je kontextová mapa(třída VasanNetwork) poskytovatelem služeb pro volání z UI části, UI má přístup jenom k metodám přístupným v kontextové mapě. Například zprostředkovává UI metody pro načtení dat do mapy, pro provedení komprese, dekomprese, generování statistik,... . Samotná třída však vykonává jenom operace, které se jí přímo týkají, všechny další funkce jenom zprostředkovává od jiných objektů:

- Metody pro načtení slov a asociací
- Uchovává a umožňuje manipulaci se slovy a asociacemi
- Umožňuje vyhledávání v kontextové mapě na základě indexu a slova

4.2.2 Neuron

Implementaci kontextové mapy jsem navrhl tak, že každé slovo v mapě je samostatný objekt, označme tento objekt termínem neuron. Při implementaci jsem původně označil chybně kontextovou mapu, jako neuronovou sít, a tak se v kódu tato třída nazývá neuron, v kodu je to bezesporu přehlednější, i když nepřesné označení. Z hlediska programátora je jednodušší si představit neuron, do kterého a z kterého vedou spojení na jiné neurony. Tento neuron v sobě obsahuje tyto vlastnosti:

- Název slova, které reprezentuje.
- Index slova v síti, jinými slovy, pořadí ve kterém byl objekt do kontextové mapy vložen. Z této vlastnosti vyplývá, že neexistují dvě slova, která by měla stejný index.
- Seznam objektů, se kterými je tento objekt spojen vazbou, tzv. asociace.
- Entropie vystupujících vazeb.

Neuron je zodpovědný za přeposílání síťových paketů do dalších neuronů při procesu komprimace. V rámci tohoto procesu si neuron přečte hodnoty, které jsou v paketu obsaženy, důležité je hlavně návazné slovo, do kterého bude paket přesměrovávat. Funkce neuronu je podobná funkci routeru v IP sítích. Po nalezení vazby do následného slova neuron přičte hodnotu této vazby k hodnotě kontrolního součtu a podle pořadí(zda je sudé, či liché) této vazby také přičte či odečte hodnotu vazby k střídavému kontrolnímu součtu.

Dalším úkolem neuronu při procesu komprimace je kontrolovat, zda již paket dosáhl řešení nebo, ne. Zda je v paketu řešení neuron pozná na základě počtu slov, která s sebou paket nese, pokud je neuron posledním neseným slovem, ukončí průchod paketu kontextovou mapu a vrátí jej do kompresní jednotky, aby jej ta mohla uložit.

Úkoly neuronu při dekompresi jsou tři:

1. Přeposílání paketů do návazných spojení.
2. Odstranění neplatných paketů.
3. Vyhodnocení paketu, zda neobsahuje hledané řešení.

4.2.2.1 Přeposílání paketů pokaždé, kdy neuron obdrží paket, musí tento paket naklonovat a rozeslat do svých výstupních spojení. Tento proces připomíná chování hubu v počítačových sítích. Rozeslání do výstupních spojení se řídí pravidlem, nejdříve se vkládá do spojení s nejvyšším ohodnocením vazby. Každému paketu navíc aktualizuje kontrolní součty a inkrementuje počet projdených neuronů.

4.2.2.2 Odstranění paketů předtím, než neuron umožní paketu vstoupit do dalšího spojení, zkontroluje, zda je paket stále platný. Paket má v sobě uložena metadata z komprese(index prvního slova, index posledního slova, dva kontrolní součty, počet slov, která byla komprimována) a zároveň si ukládá průběžné výsledky při průchodu sítí, takže si vytváří novou dvojici kontrolního součtu a zároveň si vede, kolik slov dosud procestoval. Nejdříve neuron zkontroluje, zda nebude při přeposlání paketu překročen počet slov, poté provede kontrolu, zda nebyl překročen kontrolní součet. Pokud dojde k porušení pravidel, je paket zahozen.

4.2.2.3 Vyhodnocení paketu probíhá formou porovnání metadat, se kterými byl paket do sítě vyslán, tedy zda obsahuje stejný počet slov, stejný index prvního a posledního slova a shodné kontrolní součty. Pokud je řešení nalezeno, jsou ukončena všechna hledání, viz. popis v další části popisující kodér a dekodér.

4.2.3 Kodér

Je třídou přebírající názvy zdrojového a výstupního souboru. Kodér zajišťuje především tyto funkce:

- Volba typu kompresního paketu
- Vpuštění paketu do kontextové mapy
- Zápis komprimované zprávy do výstupního souboru

4.2.4 Dekodér

Dekodér je třídou jejímž primárním úkolem je nalezení původní zprávy ze zprávy komprimované. K nalezení je třeba zajistit tři důležité funkce:

- Extrahovat typ kompresního paketu
- Nalézt počet slov obsažených v paketu
- V případě fixního paketu pak také extrakce prvního a posledního indexu slova a kontrolních součtů

5 Závěr

Algoritmus sice nepřekonává kompresní rekordy, na druhou stranu umožňuje při použití tématických kontextových map a následné kompresi zavedenými algoritmy, dosáhnout lepších kompresních výsledků, než jakých dosahují běžně komerční programy. Algoritmus popsaný v této práci je schopen pracovat pouze s textovými dokumenty, není určen pro kompresi jiných typů souborů. V příloze B je uveden návrh algoritmu, který má za cíl umožnit také kompresi jiných souborů, než-li jenom textových dokumentů.

Slabou stránkou algoritmu je jeho rychlost. Pro budoucí rozvoj algoritmu je tak důležité najít způsob, kterým více zefektivnit prohledávání kontextové mapy. V tuto chvíli je jedinou možností, jak zrychlit algoritmus, měnit parametry povolené entropie, počtu slov pro kompresní paket a dobu platnosti časového razítka paketu.

Při tvorbě této práce jsem narazil na další potenciální využití algoritmu, například zatřizování(tématické) textových dat na základě úspěšnosti komprese při různých tématických kontextech. Dalším využitím by mohlo být testování originality prací. U testování originality prací bychom nedosáhli deterministické odpovědi, ale algoritmus by mohl sloužit, jako pomocník, který by takové práce vytipoval. Princip by byl založen na vytvoření kontextové mapy, která by reprezentovala práci, o které si přejeme zjistit, zda z ní bylo opisováno. Samotné ověření by probíhalo, tak že bychom nad toutou kontextovou mapou provedli kompresi podezřelé práce a porovnali, zda komprese nedosahuje výsledků lepších, než by bylo normální rozdělení u prací, ze kterých opisováno nebylo.

Michal Vašinek

6 Reference

[1] Shannon, Claude E., *A mathematical theory of communication,* Bell System Technical Journal, vol. 27, pp. 379–423 and 623–656, 1948.

[2] Blelloch, Guy E., *Introduction to Data Compression,* Carnegie Mellon University, 2001.

[3] Salomon, David, *Data Compression: The Complete Reference*, Springer, NewYork, 2004.

[4] Cleary, J.G., Witten,I.H., *Data compression using adaptive coding and partial string matching, IEEE Transactions on Communications*, Vol. 32 (4), pp. 396–402, 1984.

[5] Moffat,A., *Implementing the PPM data compression scheme, IEEE Transactions on Communications*, Vol. 38 (11), pp. 1917–1921, 1990.

[6] McMahon, David, *Quantum Mechanics DeMystified*, McGraw-Hill, NewYork, 2005.

A Maticový zápis kontextové mapy

Každou mapu, která je vytvořena algoritmy budování kontextové mapy, lze reprezentovat čtvercovou maticí dimenze N, kde N je počet disponibilních slov. Například ve druhé verzi algoritmu lze takto vybudovat matici s dimenzí rovnou 256.

Každému slovu lze přiřadit stavový vektor $|v>$. Pro formální popis stavových vektorů budu využívat Diracovy notace, kde $|v>$ je sloupcový vektor, oproti tomu $<v|$ se myslí řádkový vektor.

A.1 Báze kontextové mapy

Báze je sada vektorů, pomocí které můžeme modelovat vektorový prostor všech řešení, která se v kontextové mapě nachází. Počet vektorů báze je dán dimenzí vektorového prostoru dostupných řešení. V případě první verze algoritmu je bází soustava stavových vektorů reprezentujících jednotlivá slova, ve druhé verzi je potom 256 stavových vektorů, reprezentujících různé kombinace bitů obsažené v 1 bytu. Každý stavový vektor obsahuje jednu jedničku na pozici, která jej reprezentuje, na ostatních pozicích jsou nuly.

Uvažme příklad, mějme vektorový prostor dimenze 4(znamená, že v mapě můžeme mít nanejvýš čtyři slova). Takovýto vektorový prostor je dán čtveřicí stavových vektorů, které společně tvoří bázi:

$$|n_1>=\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

$$|n_2>=\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

$$|n_3>=\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

$$|n_4>=\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

A.2 Skalární součin

Krátce zadefinuji také operaci skalárního součinu. V Diracově notaci se používá tento zápis:

$$< n_i|n_j >$$

Pomocí skalárního součinu si nyní můžeme zadefinovat ortonormální bázi. Vektory jsou ortonormální právě když platí:

$$< n_i|n_j >= \delta_{i,j}$$

kde $\delta_{i.j}$ je symbolem pro Kroneckerovu delta funkci.Stavové vektory tvořící bázi budou v kontextové mapě obecně ortonormální.

A.3 Tenzorový součin

Pro vytváření mapy je velmi vhodným nástrojem tenzorový součin. Ten je definován jako:

$$|n_i >< n_j| \tag{7}$$

Tenzorový součin generuje čtvercovou matici o dimenzi rovné počtu složek vektoru. Pokud se na matici popisující kontextovou mapu budeme dívat jako na strukturu reprezentující vazby mezi jednotlivými prvky, zjistíme, že řádky matice reprezentují první slovo vazby, zatímco sloupce reprezentují druhé slovo vazby. Váha vazeb je potom dána pozicí vazby podle řádku a sloupce matice.

A.4 Superpozice stavových vektorů - matice kontextové mapy

Díky vlastnosti tenzorového součinu, která nám vygeneruje matici s jedničkou na pozici označující vazbu mezi prvním a druhým slovem, můžeme takovéto tenzorové součiny sčítat a generovat tím další matici. Tato matice je charakteristická tím, že je superpozicí více po sobě jdoucích stavů. Taková matice M se dá obecně zapsat takto:

$$M_{m\times m} = \alpha|n_i >< n_{i+1}| + \beta|n_{i+1} >< n_{i+2}| + ... + \Omega|n_{m-1} >< m_m| \tag{8}$$

Koeficienty $\alpha, \beta, ..., \Omega$ jsou váhy přechodu mezi jednotlivými slovy, nebo znaky.

A.5 Dotazování na vazby - matice kontextové mapy jako operátor

Pokud již máme vygenerovánu matici kontextové mapy, můžeme ji aplikovat jako operátor na jednotlivé stavové vektory a zjistit tak váhy u námi zvoleného stavového vektoru. Je třeba ovšem důsledně rozlišit, zda jsme ve výsledném vektoru obdrželi váhy směřující k následujícím slovům, nebo vazby směřující z předchozího slova. V tomto prvním případě obdržíme po přenásobení matice cest požadovaným stavovým vektorem zleva

vektor reprezentující vazby vystupující z daného slova. Operátor se v literatuře zapisuje se stříškou nad znakem: $\hat{O}$. Obecně můžeme psát:

$$< n_i|\hat{M} =< váhy_{out}| \tag{9}$$

A.6 Dotazování se na libovolnou vazbu

Pokud hledáme jednu konkrétní vazbu, dotaz tvoříme tak, že násobíme operátor zprava a zleva příslušnými stavovými vektory. Takže pokud bychom hledali váhu vazby jdoucí ze stavu $(s-1)$ reprezentujícího například písmeno "H" do stavu (s) reprezentujícího písmeno "e', psali bychom.

$$< n_{s-1}|\hat{M}|n_s >= váha$$

A.7 Zápis kontrolního součtu

Kontrolní součet reprezentuje operaci sčítání hodnot vazeb pro různé průchody cesty v mapě. Jeho formální zápis je následující:

$$+ =< n_i|\hat{M}|n_{i+1} > + < n_{i+1}|\hat{M}|n_{i+2} > +...+ < n_{i+l-1}|\hat{M}|n_{i+l} > \tag{10}$$

Kde i je pořadový index slova v komprimované zprávě a l je počet komprimovaných slov pro jeden fixní paket.

A.8 Zápis střídavého kontrolního součtu

Střídavý kontrolní součet, je druhou kontrolní hodnotou obsaženou v síťovém paketu. Výsledná hodnota je dána postupným přičítáním a odečítáním vazeb mezi slovy. Jeho formální zápis následující:

$$\pm =< n_i|\hat{M}|n_{i+1} > + < n_{i+1}|\hat{M}|n_{i+2} > -...+ < n_{i+l-1}|\hat{M}|n_{i+l} > \tag{11}$$

B Návrh algoritmu pro kompresi pomocí kontextové mapy znaků

Druhá verze algoritmu si klade za cíl vyzkoušet odlišně sestavenou mapu slov. Termín slovo zde nahradím znakem. Jeden znak je v tomto kontextu myšleno 8 bitů, tedy 1 bajt. Algoritmus nyní bude budovat svou mapu na základě vstupu po jednotlivých znacích.

B.1 Algoritmus pro vybudování mapy znaků

Algoritmus pro budování sítě v případě znakové mapy bude pracovat obdobně jako v prvním případě.

1. Rozdělení zprávy na jednotlivé znaky tak, at'jsou seřazeny od prvního k poslednímu znaku zprávy.
2. Zpráva se prochází po jednotlivých znacích.
3. Pokud danému znaku předcházel jiný znak, vazba mezi těmito znaky se inkrementuje(v případě, že žádná vazba mezi těmito znaky dosud není, vytvoří se), pokud se před znakem žádný jiný znak nenacházel, pokračuje se dále.
4. Každý znak se uloží, aby v pořadí další znak znal znak předcházející.
5. Body 2,3 a 4 se opakují přes celý zdrojový soubor dat.

B.2 Algoritmus síťového paketu

Rozložení dat v síťovém paketu bude odlišné pro dva případy, první možností je mít fixní délku zakódované zprávy, druhým je potom možnost proměnné délky zprávy. První případ nepotřebuje ukládat délku slova. Druhý případ nabízí možnost rozšíření délky vstupní zprávy, nicméně je třeba připočítat další bity pro délku slova.

B.2.1 5 bajtový síťový paket s fixní délkou slova - základní paket

Základní paket slouží jako základní, minimální kompresní paket. Tento paket v sobě obsahuje pouze minimum metadat o komprimovaných zprávách.

- FI - index prvního slova, protože používáme znaky na místo slov, maximální index bude mít hodnotu 255, proto postačí pro jeho reprezentaci jeden bajt
- CH1 - kontrolní součet číslo 1
- CH2 - kontrolní součet číslo 2
- WL - délka slova
- FLAG - čtyři jedno-bitové flagy pro případnou optimalizaci, 2bity pro rozlišení typu paketu, další dva rezervovány pro budoucí použití

byte	1	2-3	4	5
proměnná	FI	CH1	FLAG—DPT	LI

Tabulka 3: Zápis bitů(bytů) 5 bajtového kompresního paketu

byte	1	2-3	4	5-6	7	8
proměnná	FI	CH1	WL—DPT	CH2	FLAG	LI

Tabulka 4: Zápis bitů(bytů) 8 bajtového kompresního paketu s proměnnou délkou slova

- LI - index posledního slova
- DPT - hloubka maximálního zanořezení při výberu cesty z jednoho znaku

B.2.2 8 bajtový síťový paket s proměnnou délkou slova a proměnnou délkou maximálního zanoření

Délka slova může nabývat libovolné velikosti, ale protože volba příliš veliké délky slova vede k exponenciálnímu nárůstů potenciálních výsledků. Pro potřeby algoritmu bude nejvyšší hodnotou od které se bude hledat optimální výsledek číslo 16. Takto nám pro zápis délky slova postačí 4 bity. Druhá čtveřice bitů je využita pro specifikaci zanoření do sítě.

B.2.3 7 bajtový síťový paket s fixní délkou slova a proměnnou délkou maximálního zanoření

Při použití fixní délky slova máme možnost rozhodnout co udělat s uštřenými čtyřmi bajty, buď je nechat na použití např. jako FLAG značku a ušetřit tak místo ní jeden bajt a nebo naopak můžeme rozšířit počet indikátorů.

Při kompresi si algoritmus za každou komprimovanou zprávou vykoná zpětnou kontrolu, zda se na daný výsledek standardním postupem algoritmu dá dojít v nejkratším a zároveň přijatelném čase. Síťový paket zná správný výsledek i postup, takže bude schopen rychle eliminovat nesprávné cesty.

byte	1	2	3	4	5	6	7
proměnná	FI	CH1	CH1	CH2	CH2	FLAG/DPT	LI

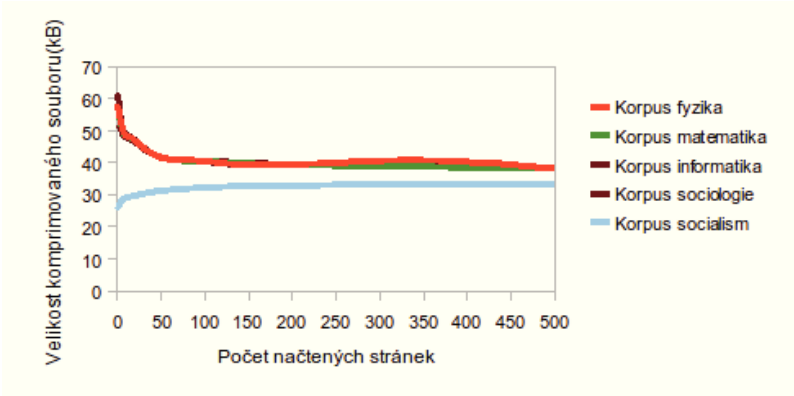
Tabulka 5: Zápis bitů(bytů) 7 bajtového kompresního paketu

C Grafy a zhodnocení pro odlišné korpusy

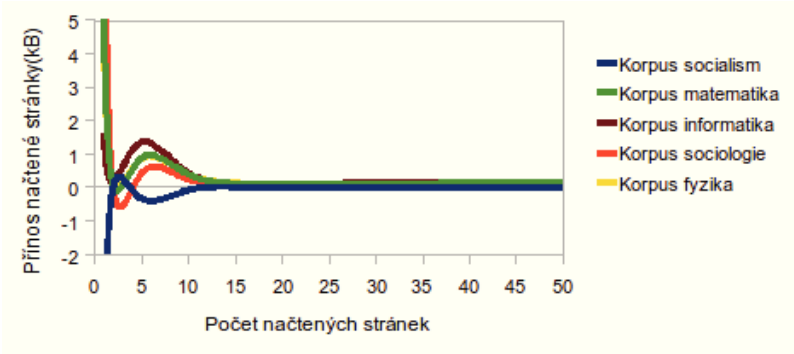
Pro otestování, zda závislosti zjištěné v kapitole 4.1.5 jsou platné všeobecně, jsem provedl několik dalších testování na odlišných tématech článků, které jsem komprimoval pomocí kontextového algoritmu. Grafy obecně potvrdily, že pro různé tématické typy komprimovaných článků jsou závislosti v souladu s analýzou z kapitoly 4.1.5. Testovanými články byly anglické články o socialismu a počasí.

Program/algoritmus	Velikost(socialism)	Velikost(počasí)	Dvojitá komprese
Původní velikost	61,6 kB	14,2 kB	
ZIP	18,8 kB	5,31 kB	
BZ2	22,0 kB	5,64 kB	
LZMA	20,2 kB	5,31 kB	
Algoritmus(Socalism-1)	26,0 kB		19,3 kB(ZIP)
Algoritmus(Weather-1)		4,44 kB	3,46 kB(ZIP)
Algoritmus(Fyzika-100)	40,1 kB	10,1 kB	6,3 kB(ZIP)
Algoritmus(Uni-500)	37,8 kB	12,4 kB	6,18 kB(ZIP)

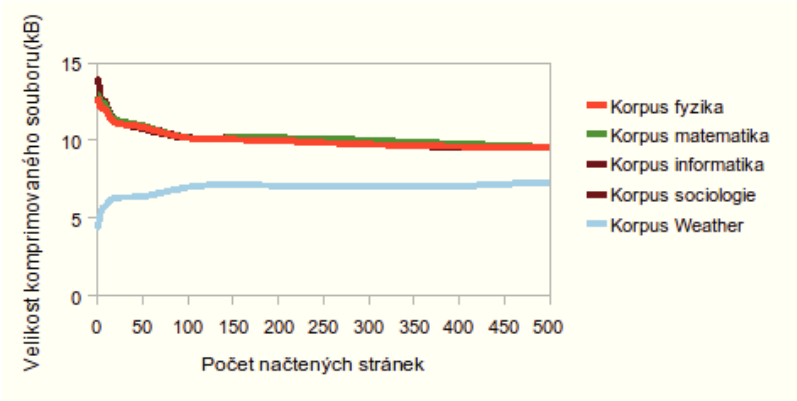
Tabulka 6: Porovnání výsledků s dalšími algoritmy



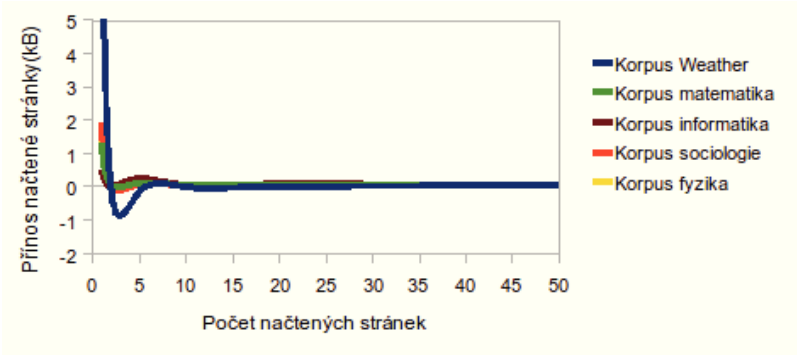
Obrázek 12: Graf závislosti velikosti výsledného souboru na počtu načtených stránek - článek socialism



Obrázek 13: Graf závislosti přínosu každé další načtené stránky - článek socialism



Obrázek 14: Graf závislosti velikosti výsledného souboru na počtu načtených stránek - článek weather



Obrázek 15: Graf závislosti přínosu každé další načtené stránky - článek weather

D DVD s implementací algoritmu